2-D Wavelet Transform Enhancement on General-Purpose Microprocessors: Memory Hierarchy and SIMD Parallelism Exploitation[1]

D. Chaver, C. Tenllado, L. Piñuel, M. Prieto and F. Tirado

Departamento de Arquitectura de Computadores y Automatica
Facultad de Ciencias Fisicas
Universidad Complutense
28040 Madrid, Spain
{dani02, tenllado, lpinuel, mpmatias, ptirado} @dacya.ucm.es

This paper addresses the implementation of a 2-D Discrete Wavelet Transform on general-purpose microprocessors, focusing on both memory hierarchy and SIMD parallelization issues. Both topics are somewhat related, since SIMD extensions are only useful if the memory hierarchy is efficiently exploited. In this work, locality has been significantly improved by means of a novel approach called pipelined computation, which complements previous techniques based on loop tiling and non-linear layouts. As experimental platforms we have employed a Pentium-III (P-III) and a Pentium-4 (P-4) microprocessor. However, our SIMD-oriented tuning has been exclusively performed at source code level. Basically, we have reordered some loops and introduced some modifications that allow automatic vectorization. Taking into account the abstraction level at which the optimizations are carried out, the speedups obtained on the investigated platforms are quite satisfactory, even though further improvement can be obtained by dropping the level of abstraction (compiler intrinsics or assembly code).

1. Introduction

Over the last few years, we have witnessed an important development in applications based on the discrete wavelet transform. The most outstanding success of this technology has been achieved in image and video coding. In fact, state-of-the-art standards such as MPEG-4 or JPEG-2000 are based on the discrete wavelet transform (DWT). It is also a valuable tool for a wide variety of applications in many different fields (image fusion [1], computer graphics [2], etc). This growing importance makes a performance analysis of this kind of transformation of great interest.

Our study focuses on general-purpose microprocessors. In these particular systems, the main aspects to be addressed are the efficient exploitation of the memory hierarchy, especially when handling large images, and how to structure the

1 This work has been supported by the Spanish research grants TIC 99-0474 and TIC 2002-750

computations to take advantage of the SIMD extensions available on modern microprocessors.

With regard to the memory hierarchy, the main problem of this transform is caused by the discrepancies between the memory access patterns of two principal components of the 2-D wavelet transform: the vertical and the horizontal filtering [2]. This difference causes one of these components to exhibit poor data locality in the straightforward implementations of the algorithm. As a consequence, the performance of this application is highly limited by the memory accesses.

The experimental platforms on which we have chosen to study the benefits of the SIMD extensions are two Intel Pentium-based PCs equipped with a P-III (SSE) and a P-4 (SSE2) processor respectively. Due to portability reasons and in order to prevent long development times, we have avoided coding at the assembly language level. Consequently, all the optimizations have been performed at the source code level. To be specific, in order to allow automatic vectorization, we have introduced some directives, which inform the compiler about pointer disambiguation and data alignment, and we have performed some code modifications such as loop transformations or variable scope changes.

This paper is organized as follows. The investigated wavelet transform and some related work are described in sections 2 and 3 respectively. The experimental environment is covered in section 4. In Section 5 we discuss the memory hierarchy optimizations, then in section 6 our automatic vectorization technique is explained and some results are presented. Finally, the paper ends with some conclusions.

2. 2-D Discrete Wavelet Transform

The discrete wavelet transform (DWT) can be efficiently performed using a pyramidal algorithm based on convolutions with Quadrature Mirror Filters (QMF). The wavelet representation of a discrete signal S can be computed by convolving S with the lowpass filter H(z) and highpass filter G(z) and downsampling the output by 2. This process decomposes the original image into two sub-bands, usually denoted as the coarse scale approximation (lower band) and the detail signal (higher band) [2].

This transform can be easily extended to multiple dimensions by using separable filters, i.e. by applying separate 1-D transforms along each dimension. In particular, we have studied the most common approach, commonly known as the square decomposition. This scheme alternates between operations on rows and columns, i.e. one stage of the 1-D DWT is applied first to all the rows of the image and then to the columns. This process is applied recursively to the quadrant containing the coarse scale approximation in both directions. In this way, the data on which computations are performed is reduced to a quarter in each step [2].

From a performance point of view, the main bottleneck of this transformation is caused by the vertical filtering (the processing of image columns) or the horizontal one (the processing of image rows), depending on whether we assume a row-major or a column-major layout for the images. In particular, all the measurements taken in our research have been obtained performing the whole wavelet decomposition using a

(9,7) tap biorthogonal filter [2]. Nevertheless, our results are qualitatively almost filter-independent.

3. Related Work

A significant amount of work on the efficient implementation of the 2-D DWT has already been done for all sorts of computer systems. Focusing on the target of this paper, i.e. general-purpose microprocessors, several optimizations aimed at improving the cache performance have been proposed in [3][4][5]. Basically, [4] and [5] investigate the benefits of traditional loop-tiling techniques, while [3] investigates the use of specific array layouts as an additional means of improving data locality.

The thesis of [3] is that row-major or column-major layouts (canonical layouts) are not advisable in many applications, since they favor the processing of data in one direction over the other. As an alternative, they studied the benefits of two non-linear layouts, known in the literature as 4D and Morton [3]. In these layouts the original mxn image is conceptually viewed as an $\lceil m/tr \rceil \times \lceil n/tc \rceil$ array of $tr \times tc$ tiles. Within each tile, a canonical (row-major or column-major) layout is employed.

The approach investigated in [4] is less aggressive. Nevertheless, they addressed the memory exploitation problem in the context of a whole application, the JPEG2000 image coding, which is more tedious to optimize than a wavelet kernel. In particular, they considered the reference implementations of the standard. By default, both implementations use a five-level wavelet decomposition with (7,9) biorthogonal filters as the intra-component transform of the coding [4]. The solution investigated by these authors, which they dubbed “aggregation”, is similar to the classical loop-tiling strategy that we have applied to the vertical filtering in [5] (the one that lacks spatial locality if a row-major layout is employed). In this scheme, instead of processing every image column all the way down in one step, which produces very low data locality (on a row-major layout, rows are aligned along cache lines), the algorithm is improved by applying the vertical filtering row by row so that the spatial locality can be more effectively exploited (see figure 1).

In [6] we have extended these previous studies with a more detailed analysis based on hardware performance counters and a study of the vectorization on an Intel P-III microprocessor.

In the present work we have explored some ideas introduced in the context of special purpose hardware for a wavelet-based image coder [7], but with quite a different goal. Instead of applying them to minimize the memory needs, which is a significant issue in mass-market consumer products due to its cost impact, we have investigated their applicability in the reduction of the computational cost, which is the main concern in general purpose computers.

As explained above, the traditional implementation of the square variant of the 2-D wavelet decomposition consists in applying the horizontal filtering to all the image lines before the column filtering starts. The algorithm proposed in [7], known as on-line computation [8], provides a significant memory saving by starting the vertical filtering as soon as a sufficient number of lines (determined by the filter length) have been horizontally filtered. As opposed to the traditional processing, which requires a

memory size of the order of the image, this line-based implementation only needs to store a minimum number of image lines. In our context, this strategy can also provide important benefits since it improves the temporal locality of the algorithm.

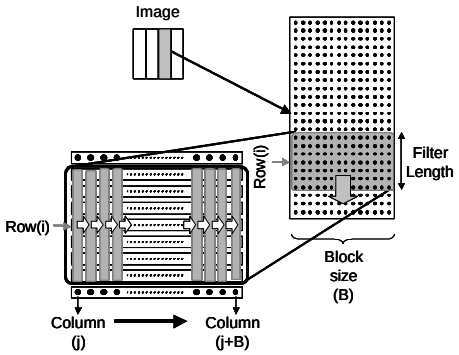

Fig. 1. Aggregation technique in the vertical filtering using a row-major layout.

4. Experimental Environment

The performance analysis presented in this paper has been carried out on two Pentium-based PCs, equipped with a P-III 866 MHz and a P-4 1,4GHz, for a more detailed description see [10]. The programs have been compiled using the Intel C/C++ Compiler for Linux (v5.0.1) and the compiler switches "-O3 -tpp6 -xK -restrict" or "-O3 -ttp7 -xW -restrict " depending on the processor (P-III or P-4). In the case of the P-III, our measurements have been made using a high-level application-programming interface, PAPI (v2.0.1) [11]. In the case of the P4 processor, the lack of high level application interfaces to access its performance counters has forced us to carry out the performance analysis using only the OS timing routines.

5. Cache Analysis and Pipelined Computation

As mentioned before, the wavelet transform poses a major hurdle for the memory hierarchy, due to the discrepancies between the memory access patterns of the two main components of the 2-D wavelet transform: the vertical and horizontal filtering [2]. Consequently, the improvement in memory hierarchy use represents the most important challenge in algorithm from a performance perspective. In this work, we have extended two of the approaches studied in [6] with a new technique that we have dubbed pipelined computation, which is conceptually similar to the on-line computation described above.

5.1. Pipelined Computation

The first approach investigated in this work combines aggregation (tiling) and the idea of online computation under a column-major layout. Both optimizations seem very complementary since the former tries to improve the spatial locality of the transform whereas the main goal of the latter is to enhance its temporal locality. In particular, since a column-major layout is employed, aggregation is applied to the horizontal filtering, the results of which, i.e. coarse approximations (A) and details (D), are stored temporally in two auxiliary buffers. In this way, when a whole column has been horizontally filtered, the vertical filtering can be applied, storing now the results (AA, AD, DA and DD) in their respective locations in the transformed image.

This strategy is conceptually similar to the online computation described above, since both approaches alternate between the horizontal and the vertical filtering. In the online computation case, the number of lines that have to be processed before applying the vertical filtering has to match up with the filter length. In this new approach, there is no start-up phase, i.e. the computation alternates between the horizontal and the vertical filtering from the beginning. Nevertheless, the memory behavior is similar in both cases. The main disadvantage of this strategy is that the size of the auxiliary buffers depends on the image dimensions. This means that those buffers become too long for large images, and consequently the temporal locality could be significantly diminished.

5.2. 4D Pipelined Computation

The second approach that we have studied combines the 4D layout with the idea of pipelined computation introduced above. It divides the processing of every tile column into three different phases (see figure 2).

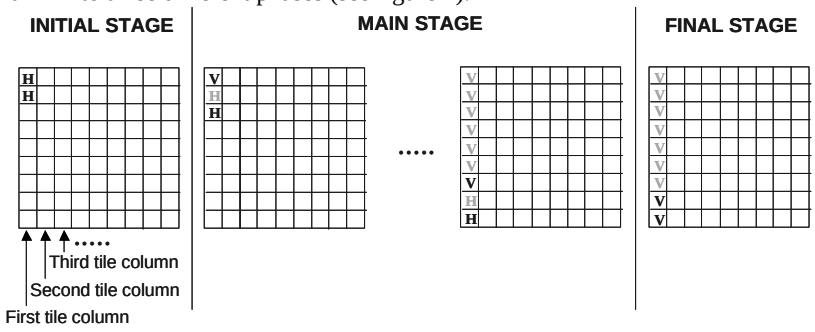

Fig. 2. Pipelined computation combined with a 4D layout.

In the initial stage, the first two blocks of the tile column are horizontally filtered (using aggregation). Then, as figures 2 and 3 illustrate, it alternates between the vertical and horizontal processing of the different blocks, concluding with a final stage where the two last blocks are vertically processed. Like the first approach, in order to perform this sequence three auxiliary blocks have to be employed.

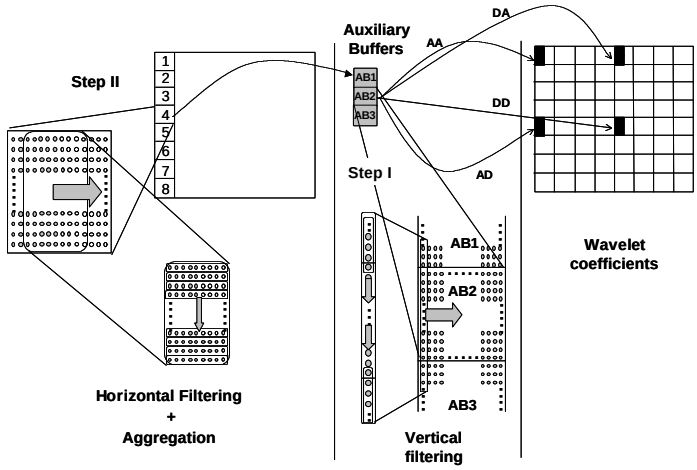

Fig. 3. Pipelined computation combined with a 4D layout (main stage processing). In the first step the auxiliary buffer AB2 is vertically filtered using neighbor coefficients from the auxiliary blocks AB1 and AB3. The second step describes the horizontal filtering of the next block, which is temporary stored in buffer AB1.

In this case, the size of the auxiliary blocks is independent of the image dimensions, and thus they could be significantly smaller than in the previous approach. Besides saving memory, now the temporal locality could be more efficiently exploited since a lower number of elements have to be horizontally filtered before the vertical one can be applied.

5.3. Performance Results

In this section we are interested in assessing the benefits of the proposed new approaches, pipelined computation and 4D pipelined computation. The results reported have been obtained using the experimental framework explained in section 4.

The performance results for the four codes under study, when processing an image of 8192^2 pixels, are displayed in figures 4 and 5. We should firstly remark that the 4D versions present a significantly lower number of misses than the other implementations. This poor behavior of the aggregated and the pipelined versions is mainly caused by the number of conflict misses involved in the computations of the horizontal filters. Since these approaches use a column-major layout, these computations require access to non-continuous memory blocks with a stride that depends on image height. Given that the image sizes considered in this research are to the power of 2, above a certain height these blocks are mapped into the same cache set, provoking an important number of misses (we have employed a 9-tap filter whereas both processors only have 4 lines per set in the L1 data cache and 8 lines per set in the L2). Under a 4D layout the mapping of these blocks does not dependent on the image height but on the tile height, which is much lower. Consequently, the number of conflict misses is significantly reduced.

Besides this difference between linear and non-linear layouts, we observe that both versions of the pipelined computation significantly reduce the L2 misses due to their better temporal locality (the pipelined and the 4D pipelined outperform the aggregation and the 4D approaches by about 14% and 42% respectively). This improvement in the L2 exploitation translates into a significant performance gain (as can be seen, the execution time is strongly related to the L2 behavior), especially in the case of the 4D layout, where the pipelined implementation achieves around a 32% speedup on both the P-III and P-4 processors.

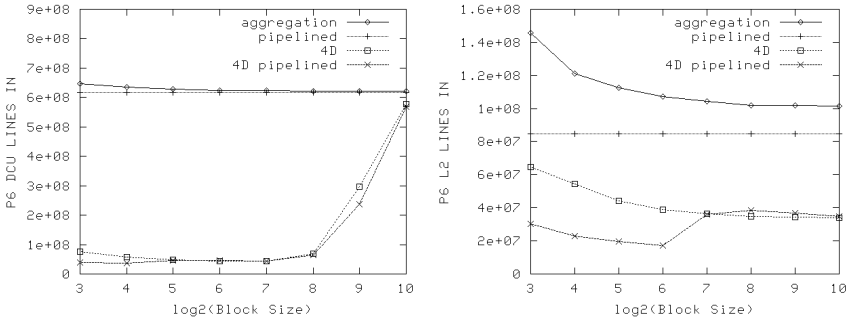

Fig. 4. L1 and L2 cache behavior for an 8192^2 pixels image (P-III). The memory hierarchy behavior has been monitored through the "DCU LINES IN" and "L2 LINES IN" events, which represent the number of lines that have been allocated in the L1 data cache, and the number of L2 allocated lines respectively.

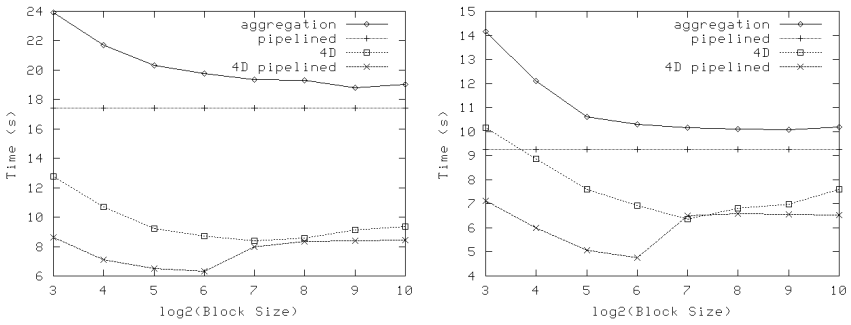

Fig. 5. Execution time for an 8192^2 pixels image on the P-III (left) and the P-4 (right).

Results for a 4096^2 pixels image are shown in figures 6 and 7. Now, we observe a quite different behavior of the L2 cache since, unlike the previous case, the number of conflict misses in the aggregation and pipelined approaches are not so significant. As a consequence, the results of aggregation and 4D are similar. Nevertheless, the benefits of the pipelined implementations are as significant as for the 8192^2 pixels image. In this case, given that the L2 conflict misses problem has disappeared, the pipelined even outperforms the 4D approach.

Table 1 summarizes the results obtained for different image sizes on both platforms. The approach that has always exhibited the more efficient memory access pattern and the higher performance has been the 4D pipelined. The observed speedups over the 4D approach range from a factor of 30% up to 40% on the P-III (independently of the image size), whereas on the P-4 it grows with the image size, from 30% up to 50%. Nevertheless, the simple pipelined version achieves competitive results for small image sizes, since in these cases it also captures the temporal locality.

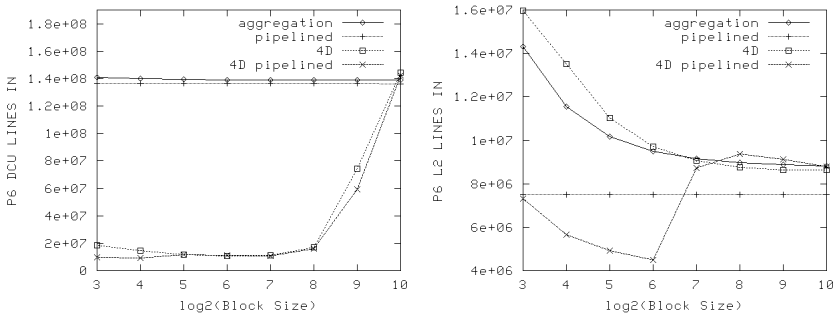

Fig. 6. L1 and L2 cache behavior for a 4096^2 pixels image (P-III).

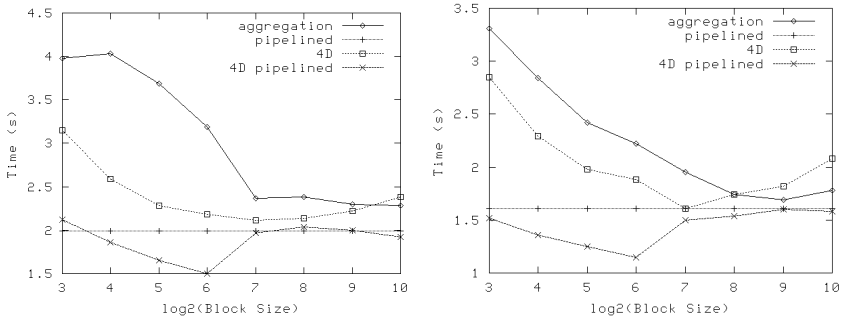

Fig. 7. Execution time for an 4096^2 pixels image on the P-III (left) and the P-4 (right).

Table 1. Percentage of Speedup achieved with the investigated optimization.

Image Size		512^2	1024^2	2048^2	4096^2	8192^2
P-III	**Pipelined over Aggregation**	17.2	15.2	16.6	14.0	7.9
	4D Pipelined over 4D	30.7	31.7	31.0	40.6	32.5
	Pipelined over Aggregation	9.0	8.7	5.4	6.2	9
P-4	**4D Pipelined over 4D**	47.0	48.5	40.7	35.6	32.6
	4D Pipelined over Pipelined	29.4	33.8	37.0	39.1	194.7
	4D Pipelined over Pipelined	11.5	20.2	21.4	32.7	276.5

6. SIMD Optimization

Previous research on the parallel wavelet transform has been concentrated on special purpose hardware and out-of-date SIMD architectures [13][14][15]. Work on general-purpose multiprocessor systems includes [5] and [16], where different parallel strategies for the 2-D wavelet transform were compared on the SGI Origin 2000, the IBM SP2 and the Fujitsu VPP3000 systems respectively. In [17] a highly parallel wavelet transform is presented but at the cost of changing the wavelet transform semantic. Other work includes [18], where several strategies for the wavelet-packet decomposition are studied.

We have focused our research on the potential benefits of Single Instruction Multiple Data (SIMD) extensions. Among related work, we can mention [19], where an assembly language vectorization of real and complex FIR filters is introduced based on Intel SSE. However, our main interest is to assess whether it is possible to take advantage of such extensions to exploit the data parallelism available in the wavelet transform, in a filter-independent way and avoiding low level programming. Therefore, the same strategy introduced in [6] has been followed to carry out the vectorization. The present research extends this previous work by adding the analysis of the 4D Pipelined Computation introduced above. Furthermore, performance results on an Intel P-4 processor are also presented. As in [6], we have first studied how to automatically vectorize the horizontal filtering before addressing the vectorization of the whole transform.

6.1. Guided Automatic Vectorization of the Horizontal Filtering

From a programmer's point of view, the most suitable way to exploit SIMD extensions is automatic vectorization since it avoids low level coding techniques, which are platform dependent. Nevertheless, loops must fulfill some requirements in order to be automatically vectorized, and in most practical cases both code modifications and guided compilation are usually necessary. In particular, the Intel compiler [9] can only vectorize simple loop structures. Primarily, only loops with simple array index manipulation (i.e. unit increment) and which iterate over contiguous memory locations are considered (thus avoiding non-contiguous accesses to vector elements). Obviously, only inner loops can be vectorized. In addition, global variables must be avoided since they inhibit vectorization. Finally, if pointers are employed inside the loop, pointer disambiguation is mandatory (this must be done by hand using compiler directives). Considering these restrictions, under a column-major layout only the "aggregated" horizontal filter can be automatically vectorized. For more details see [10].

6.2. Vectorization of the Whole Transform

The automatic vectorization is relatively straightforward in the horizontal filtering, since image elements are stored contiguously in columns. However, in order to apply the same technique to the vertical filtering, we needed the elements to be stored

contiguously in rows. Guided by [19], where an assembly language vectorization of FIR filters is introduced, we have tried to force the compiler to automatically vectorize the filter itself. However, this approach is not possible without dropping the abstraction level, which is not the goal pursued in this work.

The approach investigated in [6] consists in applying a block transposition of the resultant wavelet coefficients from the horizontal filtering, so that vectorization can be performed automatically. In this research, we have also followed this vectorization strategy, which is graphically described in figure 8. As can be seen, the results of the horizontal filtering are temporarily stored in a new auxiliary block before performing the required block transposition. In order to reduce the overheads involved in this operation, an optimized 4x4 matrix transposition based on the Intel compiler predefined _MM_TRANSPOSE4_PS macro [9] has been employed.

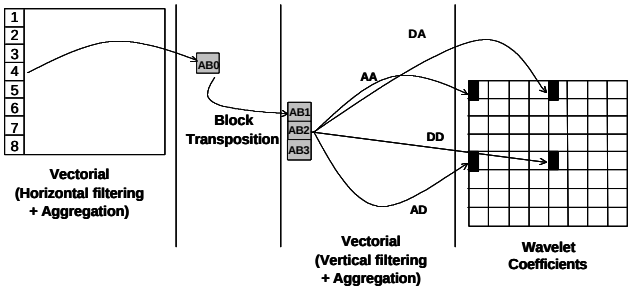

Fig. 8. Vectorial 4D pipelined computation.

6.3 Experimental Results

We have restricted this section to the 4D pipelined due to its superior performance. The improvements achieved when vectorization is performed are summarized in table 2. The vectorized horizontal filtering beats the running times of the scalar version for all image sizes. On average, the improvement is better on the P-III (around 35%) than on the P-4 (around 25%). The reason behind this behavior is that for the former processor, the Intel compiler also introduces data-prefetch when vectorization is enabled [6]. Therefore, the 35% achieved on the P-III is due to both data prefetching and SIMD extensions, whereas on the P-4 the 25% is only caused by the SIMD parallelism (the compiler does not introduce data-prefetch instructions in the P-4).

Further improvements are obtained when both filterings are vectorized. On average, the speedup for the P-III only grows to 45% due to the overheads associated with the block transposition. In fact, for a 512^2-image size it is better not to apply the vectorized vertical filtering. However, in the P-4, where we have observed lower transposition overheads (percentagewise), the average speedup extends to 70%. In fact, the improvement grows with the image size, reaching a significant 90% for the 8192^2-image since the impact of the block transposition becomes insignificant.

Table 2. Percetange of Speedup achieved by the vectorized versions over the non-vectorized 4D pipelined for the targer platforms.

Image Sizes		512^2	1024^2	2048^2	4096^2	8192^2
P-III	**Horiz.**	44.4	35.1	34.8	26.0	31.2
	Horiz+Vert.	36.8	42.5	45.5	45.6	52.9
P-4	**Horiz.**	30.7	21.4	21.3	27.7	30.8
	Horiz+Vert.	54.5	65.8	71.1	85.4	90.0

7. Conclusions

In this paper we have introduced a novel approach to optimizing the computation of the 2D DWT, called 4D pipelined computation, which significantly improves on the performance of previous approaches. The main conclusions can be summarized as follows:

1. Focusing on the memory hierarchy exploitation, the pipelined computation significantly reduces the L2 misses due to its better temporal locality. This improvement translates into a significant performance gain, especially in the case of the 4D layout, where the pipelined implementation achieves around a 30-50% of speedup on both the P-III and P-4 processors. Furthermore, we should highlight that this technique provides significant benefits not only for large image size but also for a wide range of image sizes.
2. We have introduced a novel approach to structuring the computation of the wavelet coefficients that allows automatic vectorization, which is independent of the filter size and the computing platforms (assuming that similar SIMD extensions are available).
3. In order to apply the vectorization to both filterings (horizontal and vertical) a block transposition is required. However, the performance gain achieved through vectorization by far compensated the transposition overhead, especially on the P-4. The overall speedup obtained with regard to the previous 4D and aggregation approaches is on average around 2 for the P-III and 2.5 for the P-4.

Finally, we should remark that further improvements of the 2-D wavelet transform can be achieved by introducing a lifting-based scheme [2], since this reduces the computational complexity of the algorithm. Preliminary results obtained with this approach are qualitatively comparable to their convolution-based counterpart. However, we should note that in order to apply the investigated optimizations, a similar memory waste is required. In other words, our approach also provides an efficient way to exploit the memory hierarchy and the SIMD parallelism for lifting, but at the cost of ignoring its memory saving characteristics. Taking into account the abstraction level at which the optimizations are carried out, the speedups obtained on the investigated platforms are quite satisfactory, even though further improvement can be obtained by dropping the level of abstraction (compiler intrinsics or assembly code).

References

[1] Z. Zhang and R. S. Blum. A Categorization of Multiscale-Decomposition-Based Image Fusion Schemes with a Performance Study for a Digital Camera Application. Proceeding of the IEEE, Vol. 87(8): 1315-1325, August 1999.
[2] E. J. Stollnitz, T. D. DeRose and D. H. Salesin. Wavelets for Computer Graphics: Theory and Applications. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling, Morgan Kaufmann Publishers, Inc. San Francisco, CA, 1996.
[3] S. Chatterjee, V. V. Jain, A. R. Lebeck, S. Mundhra and M. Thottethodi. Nonlinear Array Layouts for Hierarchical Memory Systems. Proceedings of 1999 ACM International Conference on Supercomputing, pp. 444-453, Rhodes, Greece, June 1999.
[4] P. Meerwald, R. Norcen, and A. Uhl. Cache issues with JPEG2000 wavelet lifting. In proceedings of 2002 Visual Communications and Image Processing (VCIP'02), volume 4671 of SPIE Proceedings, San Jose, CA, USA, January 2002.
[5] D. Chaver, M. Prieto, L. Piñuel, F. Tirado. Parallel Wavelet Transform for Large Scale Image Processing. Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'2002). Florida, USA, April 2002.
[6] D. Chaver, C. Tenllado, L. Piñuel, M. Prieto and F. Tirado. Wavelet Transform for Large Scale Image Processing on Modern Microprocessors. To be published in the proceedings of Vecpar 2002, Porto, Portugal, June, 2002.
[7] C. Chrysafis and A. Ortega. Line Based Reduced Memory Wavelet Image Compression. IEEE Trans. on Image Processing, Vol 9, No 3, pp. 378-389, March 2000.
[8] M. Vishwanath, The recursive pyramid algorithm for the discrete wavelet transform. IEEE Trans. Signal Processing, vol. 42, pp. 673-676, March 1994.
[9] Intel Corp. Intel C/C++ Compiler for Linux. Information available at http://www.intel.com/software/products/compilers/c50/linux
[10] D. Chaver, C. Tenllado, L. Piñuel, M. Prieto and F. Tirado. Vectorizing the Wavelet Transform on the Intel Pentium-III and Pentium-4 Microprocessors. Technical Report 02-001. Dept. of Computer Architecture. Complutense University, 2002.
[11] K. London, J. Dongarra, S. Moore, P. Mucci, K. Seymour and T. Spencer. End-user Tools for Application Performance Analysis, Using Hardware Counters. Presented at International Conference on Parallel and Distributed Computing Systems. August 2001.
[13] C. Chakrabarti and C. Mumford. Efficient realizations of encoders and decoders based on the 2-D discrete wavelet transforms. IEEE Trans. VLSI Syst., pp. 289-298, September 1999.
[14] T. Denk and K. Parhi. LSI Architectures for Lattice Structure Based Orthonormal Discrete Wavelet Transforms. IEEE Trans. Circuits and Systems, vol. 44, pp. 129-132, February 1997.
[15] M. Holmström. Parallelizing the fast wavelet transform. Parallel Computing, 11(21): 1837-1848, April 1995.
[16] O.M. Nielsen and M. Hegland. Parallel Performance of Fast Wavelet Transform. International Journal of High Speed Computing, 11 (1): 55-73, June 2000.
[17] L. Yang and M. Misra. Coarse-Grained Parallel Algorithms for Multi-Dimensional Wavelet Transforms. The journal of Supercomputing 11:1-22, 1997.
[18] M. Feil and A. Uhl. Multicomputer algorithms for wavelet packet image decomposition. Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'2000), pages 793-798, Cancun, Mexico, 2000. IEEE Computer Society.
[19] Intel Corp. Real and Complex FIR Filter Using Streaming SIMD Extensions. Intel Application Note AP-809. Available at http://developer.intel.com.